



FIG. 1

Structure of a server

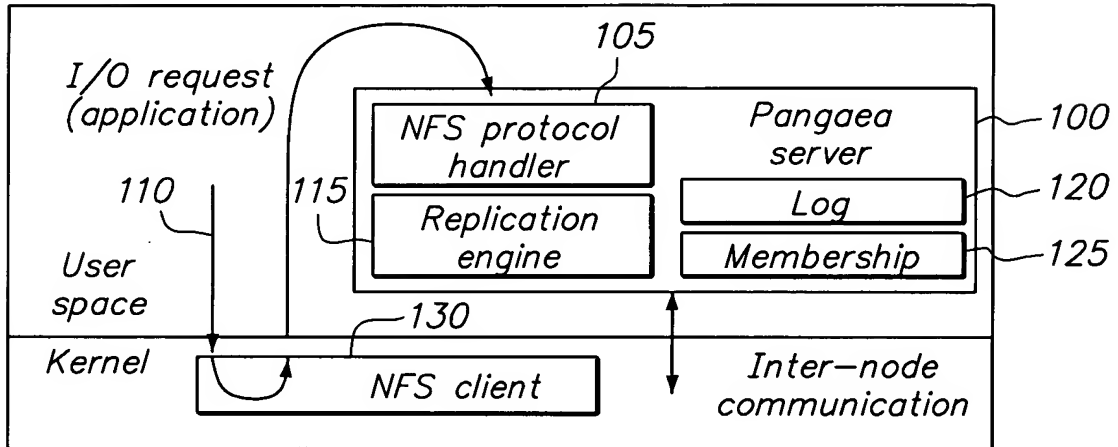
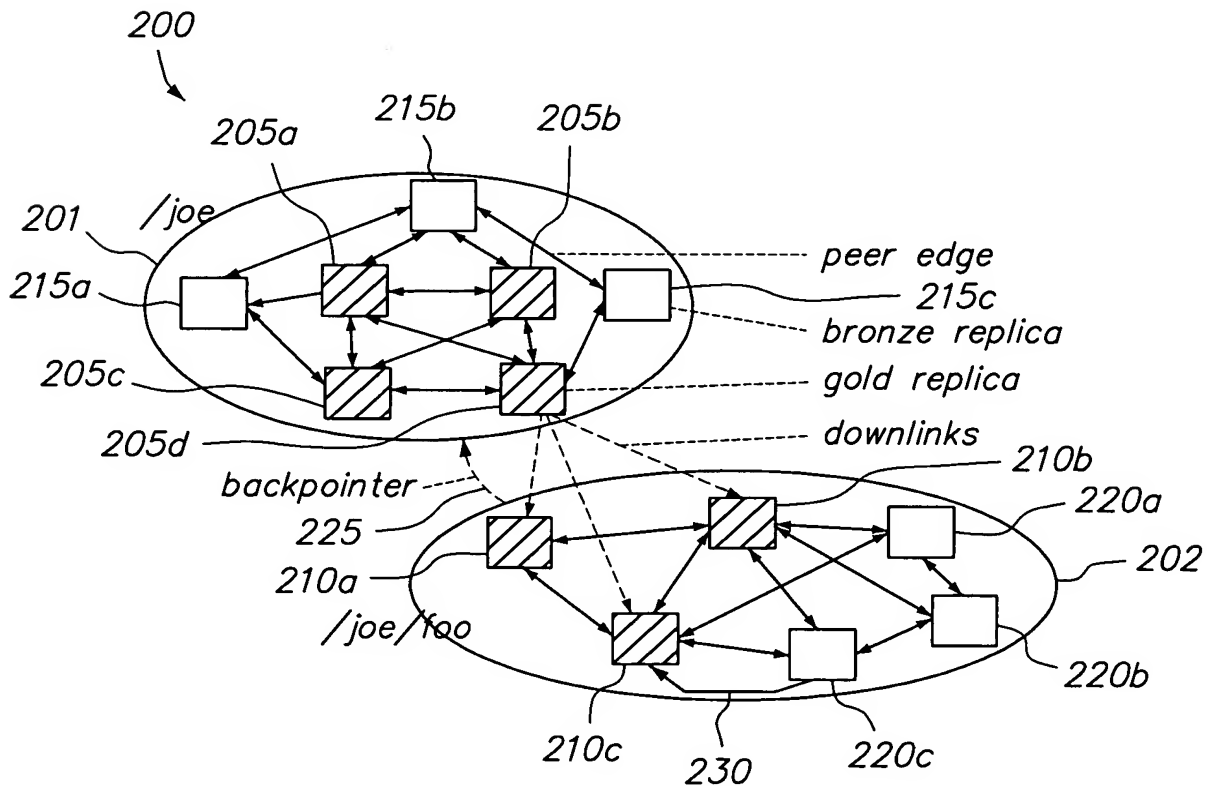


FIG. 2



Key attributes of a replica

```

struct Replica
  fid: FileID
  ts: Timestamp
  vv: VersionVector
  goldPeers: Set(NodeID)
  peers: Set (NodeID)
  backptrs: Set(FileID, String)
  ...
end
struct DirEntry
  fname: String
  fid: FileID
  downlinks: Set (NodeID)
  ts: Timestamp
end
  
```

// 96 bit globally unique file ID
 // Pair of hphysical clock, IP address
 // Maps IP address 7: Timestamp
 // Set of IP addresses
 // Pair of hdirID, fnamei

FIG. 3

```

proc UpdateReplica
   $r_2$ : Replica // New replica contents.
preconditions:
   $\forall \langle pfid, fname \rangle: r_2.bptrs \bullet pfid \in \text{dom}(DISK)$ 
  and  $IsLive(r_2) \Rightarrow IsLive(DISK(pfid))$ 
postconditions:
   $r_2.bptr \neq \{\}$  (3)

```

FIG. 4

- “ $\langle val_1, \dots, val_n \rangle$ ” represents a tuple of values.
- “ \mathbb{P} **type**” represents a (possibly empty) set of **type**.
“ \mathbb{P}_1 **type**” represents a nonempty set of **type**.
“**Key** \mapsto **Val**” represents a one-to-many mapping from type **Key** to **Val**.
- “ $\text{dom}(F)$ ” returns the domain of function (or mapping) F , and “ $\text{ran}(F)$ ” returns the range of F . For instance,

$$\begin{aligned} \text{dom}(\{1 \mapsto 3, 2 \mapsto 8, 4 \mapsto 3\}) &= \{1, 2, 4\}, \\ \text{ran}(\{1 \mapsto 3, 2 \mapsto 8, 4 \mapsto 3\}) &= \{3, 8\}. \end{aligned}$$

- “ $X \oplus Y$ ” substitutes a part of mapping X by Y . E.g.,

$$\begin{aligned} \{1 \mapsto 3, 2 \mapsto 1\} \oplus \{1 \mapsto 5, 3 \mapsto 4\} \\ = \{1 \mapsto 5, 2 \mapsto 1, 3 \mapsto 4\}. \end{aligned}$$

- “ $X \triangleleft Y$ ” means function-domain restriction. E.g.,

$$\{2\} \triangleleft \{1 \mapsto 3, 2 \mapsto 8, 4 \mapsto 6\} = \{1 \mapsto 3, 4 \mapsto 6\}.$$

- “ $\forall var: \text{set} \bullet \text{expr}$ ” means that expr holds for var in set .
E.g.,

$$\forall n: \{11, 13, 17\} \bullet \text{IsPrime}(n).$$

- “ $\diamond \text{expr}$ ” means that expr holds eventually.
- “ $\{var: \text{set} \bullet \text{expr}\}$ ” means set comprehension. E.g.,

$$\{x: \{1, 2, 3\} \bullet x^2\} = \{1, 4, 9\}.$$

FIG. 5

Replacement Sheet

4/22

```

type Replica = record
  fid(1) : FileID
  peers(2) :  $\mathbb{P}$  NodeID
  gpeers(3) :  $\mathbb{P}_1$  NodeID
  bptrs(4) :  $\mathbb{P}$  Backptr
  deadBptr(5) : Backptr
  ts(6) : Timestamp
type RegularReplica inherits Replica =
  contents(7) : Data
Invariants:
   $\neg \text{IsLive}(r) \Rightarrow \text{contents} = \{\}$ 
type DirReplica inherits Replica =
  ents(8) :  $\langle \text{FileID}, \text{String} \rangle \rightarrow \text{DEntry}$ 
Invariants:
   $\neg \text{IsLive}(r) \Rightarrow \text{ents} = \{\}$ 
type Backptr =  $\langle \text{FileID}, \text{String} \rangle$ 
type Dentry = record
  valid(9) : bool
  ts(10) : Timestamp
  gpeers(11) :  $\mathbb{P}_1$  NodeID
proc IsLive(r)
  return r is the root or r.bptrs  $\neq \{\}$ 
  
```

FIG. 6

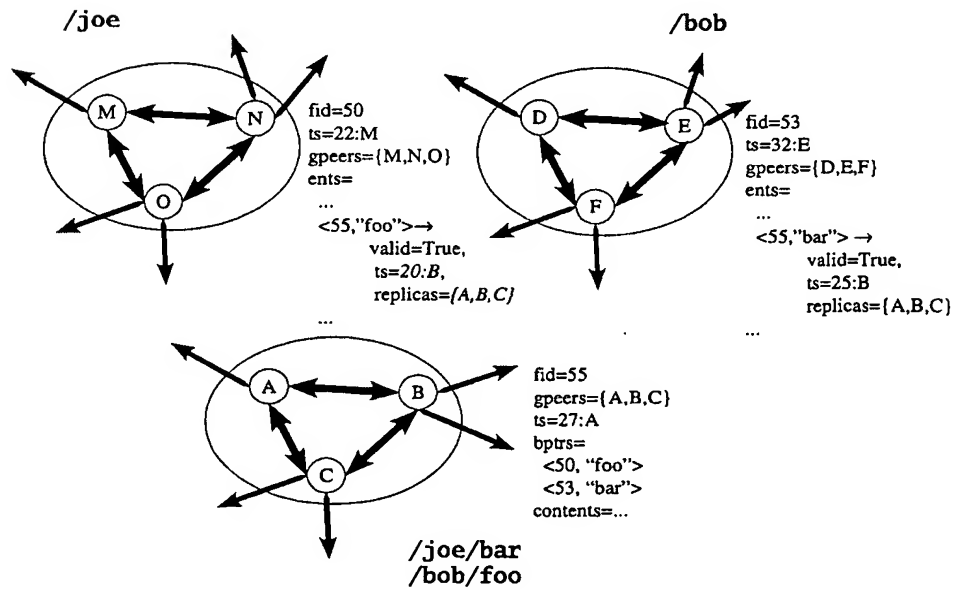


FIG. 7

Replacement Sheet

5/22

DISK: FileID \rightarrow Replica
CLOG: FileID $\rightarrow \mathbb{P}_1$ NodeID
ULOG: FileID $\rightarrow \mathbb{P}$ Backptr
Invariants::
// Updates are only for existing replicas.
 $\text{dom}(\text{CLOG}) \cup \text{dom}(\text{ULOG}) \subseteq \text{dom}(\text{DISK})$ ⁽¹²⁾

FIG. 8

```
proc Create
  d: DirReplica // The local replica of the parent directory.
  fname: string // The name of the new file in d
  gpeers:  $\mathbb{P}_1$  NodeID // The placement of the replicas of the file.
preconditions:
  IsLive(d) (13)

  r  $\leftarrow$  Newreplica()
  r.fid  $\leftarrow$  Newfileid()
  r.gpeers  $\leftarrow$  gpeers
  r.ts  $\leftarrow$  Newtimestamp()
  r.peers  $\leftarrow$  {}
  r.bptrs  $\leftarrow$  {<d.fid, fname>}
  r.contents  $\leftarrow$  None
  UpdateReplica(r)
```

FIG. 9

```
proc Unlink
  f: Replica // The file to be unlinked.
  d: DirReplica // The directory the file belongs to.
  fname: string // f's name in d.
preconditions:
  IsLive(d)
  f is a directory  $\Rightarrow$  f.ents = {}
  <d.fid, fname>  $\in$  f.bptrs

  f'  $\leftarrow$  Deepcopy(f)
  f'.bptrs  $\leftarrow$  f.bptrs \ {<d.fid, fname>}
  if f'.bptrs = {} then
    f'.deadBptr  $\leftarrow$  <d.fid, fname>
  f'.ts  $\leftarrow$  Newtimestamp()
  UpdateReplica(f')
```

FIG. 10

Replacement Sheet

6/22

```
proc Hardlink
  f: RegularReplica // The replica of the file.
  d: DirReplica // The directory to which f will be linked to.
  fname: string // The filename within d.
preconditions:
  IsLive(d)



---


  f' ← Deepcopy(f)
  f'.bptrs ← f.bptrs ∪ {⟨d.fid, fname⟩}
  f'.ts ← Newtimestamp()
  UpdateReplica(f')
```

FIG. 11

```
proc Rename
  f: Replica // The file to be moved.
  dF: DirReplica // The origin dir.
  dT: DirReplica // The destination dir.
  fnameF: string // The filename in dF
  fnameT: string // The filename in dT
preconditions:
  IsLive(dF) and IsLive(dT)
  ⟨dF.fid, fnameF⟩ ∈ f.bptrs



---


  f' ← Deepcopy(f)
  f'.bptrs ← f.bptrs \ {⟨dF.fid, fnameF⟩} ∪ {⟨dT.fid, fnameT⟩}
  f'.ts ← Newtimestamp()
  UpdateReplica(f')
```

FIG. 12

```
proc Write
  f: RegularReplica
  newcontents: Data



---


  f' ← Deepcopy(f)
  f'.contents ← newcontents
  f'.ts ← Newtimestamp()
  UpdateReplica(f')
```

FIG. 13

Replacement Sheet

7/22

```

proc UpdateReplica
   $r_2$ : Replica // New replica contents.
preconditions:
  // All parent directories are stored locally.
  // Moreover, if  $r_2$  is live, then parent must also be live.
   $\forall \langle pfid, fname \rangle: r_2.bptrs \bullet pfid \in \text{dom}(\text{DISK})$ 
  and  $\text{IsLive}(r_2) \Rightarrow \text{IsLive}(\text{DISK}(pfid))$  (14)

if  $r_2.fid \notin \text{dom}(\text{DISK})$  then
  // The replica isn't locally stored yet.
   $\text{DISK} \leftarrow \text{DISK} \cup \{ r_2.fid \mapsto r_2 \}$ 
  IssueCupdate( $r_2$ )
  return

 $r_1 \leftarrow \text{DISK}(r_2.fid)$ 

if File is regular then
  Do some application-specific stuff.
  We can potentially use version vectors here.
else
  // Union dir entries, taking ones with newer timestamps on conflict.
  for  $(key \mapsto e) \in r_2.ents$ 
    if  $key \notin \text{dom}(r_1.ents)$  or  $r_1.ents(key).ts < e.ts$ 
       $r_1.ents \leftarrow r_1.ents \oplus \{ key \mapsto e \}$ 
    for each added or deleted entry  $\langle fid, fname \rangle$  in  $r_1.ents$ 
      // Entry  $\langle fid, fname \rangle$  is potentially inconsistent. Fix up later.
      if  $fid \in \text{dom}(\text{DISK})$  then
        IssueUupdate( $\text{DISK}(fid), \{ \}$ ) (15)

if  $r_2.ts > r_1.ts$  then (16)
  // The file's attributes are to be updated.
   $r_1.ts \leftarrow r_2.ts$ 
  if  $r_1.gpeers \neq r_2.gpeers$  then
     $r_1.gpeers \leftarrow r_2.gpeers$ 
    // When the replica's gold-peer set changes, I must reflect the
    // change to the parent dir entry.
    IssueUupdate( $r, \{ \}$ )

  // Resolve potential conflicts on back pointers
  if  $r_1.bptrs \neq r_2.bptrs$  or  $r_1.deadBptr \neq r_2.deadBptr$  then
    IssueUupdate( $r_1, r_1.bptrs \setminus r_2.bptrs$ ) (17)
     $r_1.bptrs \leftarrow r_2.bptrs$ 
     $r_1.deadBptr \leftarrow r_2.deadBptr$ 

  // If the last link to the replica is gone, erase the contents.
  if  $\neg \text{IsLive}(r_1)$  then
    if  $r_1$  is a regular file then
       $r_1.contents \leftarrow \text{None}$ 
    else
      for  $e \in r_1.ents \bullet e.valid$ 
        IssueUupdate( $\text{DISK}(e.fid), \{ \}$ ) (18)
       $r_1.ents \leftarrow \{ \}$ 

if Any of  $r_1$ 's attributes has changed then
  IssueCupdate( $r_1$ ) (19)

```

FIG. 14

8/22

FIG. 15

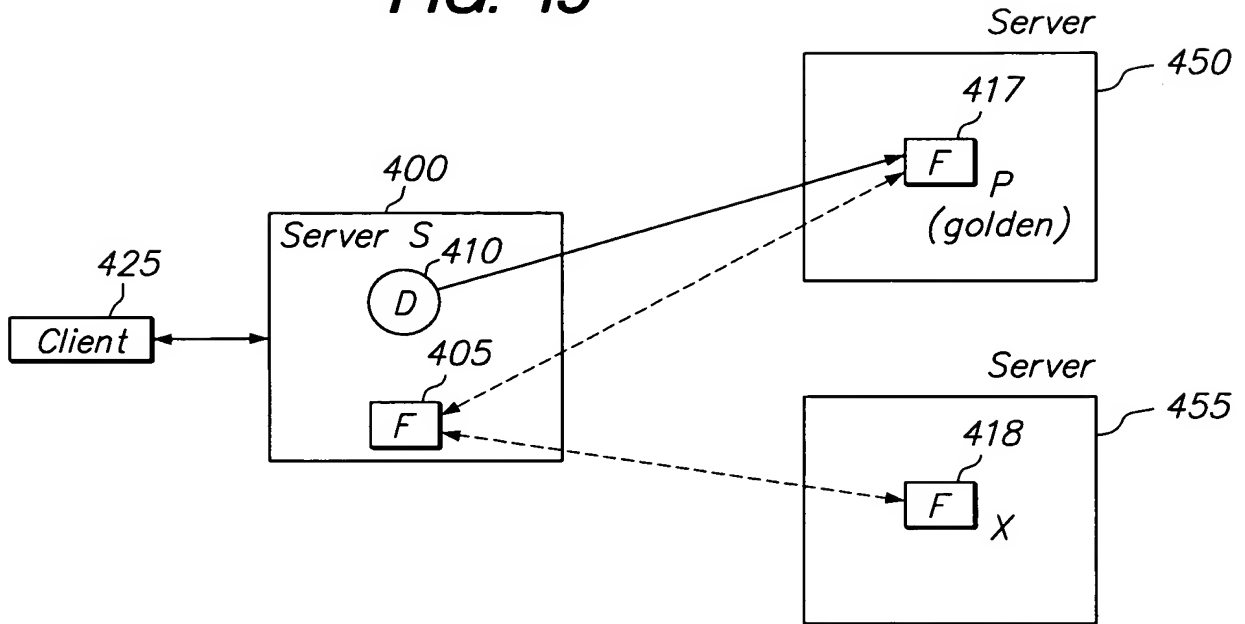
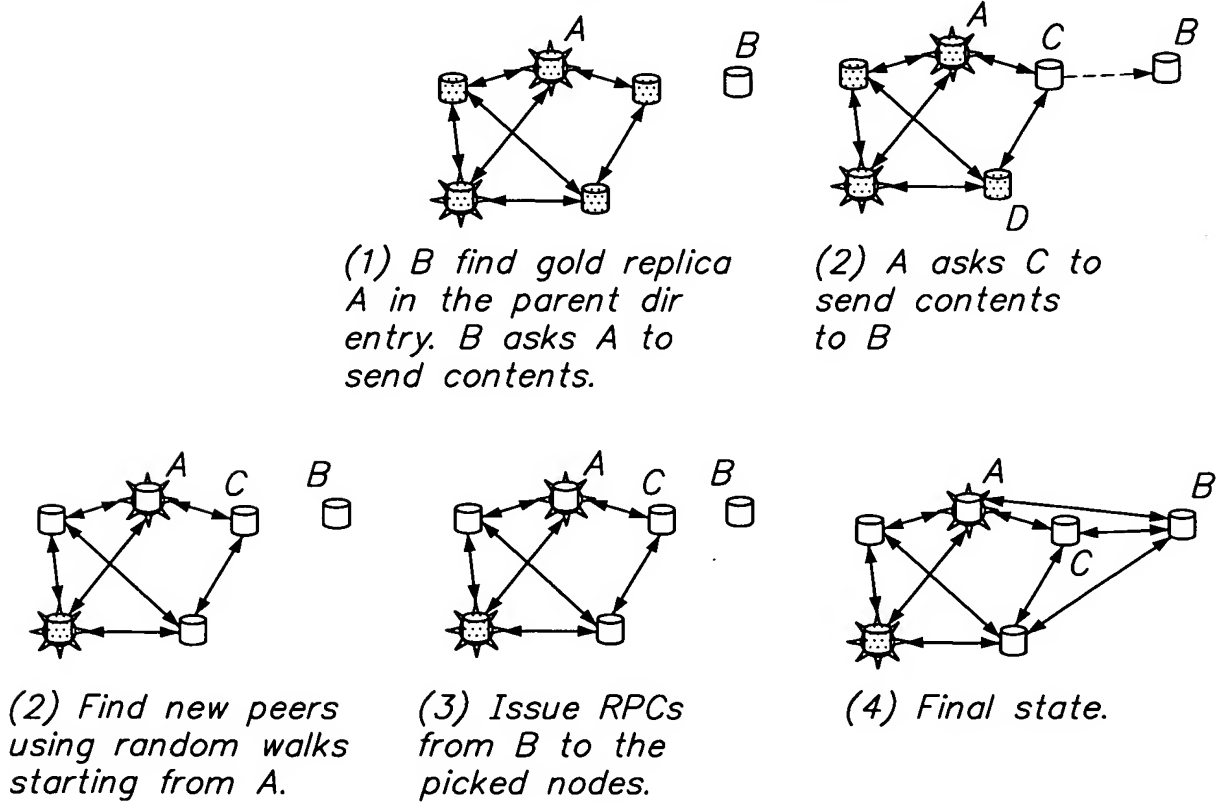


FIG. 17

Adding or deleting a bronze replica



Replacement Sheet

9/22

```
Protocol for adding a replica.

Constants:
    M: Number of neighbors per replica.
    MAXHOPS: The number of hops per a random walk (the
usual value is 3)

#
# AddReplica is the main procedure that adds
# a replica of file F on the executing node.
#
AddReplica(F, G)
    G: the set of gold replicas of F.
    (G is obtained by looking up the parent directory)

    g = Pick a random live node in G.
    Send to g, "CreateReplica(F, myself)"
    Wait for the contents to arrive.
    Store contents and reply the client.

    r = find the replica of F.
    Send to g, "StartRandomWalk(F, myself)"
    Wait for the set of neighbors N to arrive.
    for n in N:
        Add edge to n in r.
        Send to n, "AddEdge(F, myself)"

SendReplicaContents(F, Sender):
    F: the ID of the file
    Sender: the node requesting replica creation.

    r = find the replica of F
    n = pick the replica closest to Sender among
graphneighbors of r.
    Send to n, "SendReplicaContents(F, Sender)"

SendReplicaContents(F, Sender):
    F: the ID of the file
    Sender: the node requesting replica creation.

    r = find the replica of F
    Send r to Sender.

StartRandomWalk(F, Sender):
    F: the ID of the file
    Sender: the node requesting replica creation.
```

FIG. 16A

Replacement Sheet

10/22

```
r = find the replica of F
N = {}
for i = 0 to M-2:
    n = pick random graph neighbor in r.
    Send to n, "DoRandomWalk(F, 0, myself)"
    Receive nodeid from n.
    Add nodeid to N.
    Send N to Sender.

DoRandomWalk(F, hops, prevHopNode):
    F: the ID of the file
    hops: the number of hops made so far.

    if hops == MAXHOPS
        Send myself to prevHopNode
    else
        r = find the replica of F.
        n = pick random graph neighbor in r
        Send to n, "DoRandomWalk(F, hops + 1, myself)"
        Receive nodeid from n.
        Send nodeid to prevHopNode

AddEdge(F, peer):
    F: the ID of the file
    peer: the node to span edge to
    r = find the replica of F
    Add edge to peer in r
```

FIG. 16B

11/22

proc IssueCupdate <i>r</i> : Replica // The replica of the file being updated. <hr/> $CLOG(r.fid) \leftarrow r.gpeers \cup r.peers$
proc PropagateCupdate // Runs periodically in the background <hr/> for ($fid \mapsto targets$) $\in CLOG$ <i>r</i> $\leftarrow DISK(fid)$ // See (12). // Send the location of the parent dirs so that the target can // replicate them to ensure name-space containment. $pDirs \leftarrow \{p: r.bptrs \bullet \langle p.fid, DISK(p.fid).gpeers \rangle\}$ // See (14). for $n \in targets$ send $\langle CUPDATE, r, pDirs \rangle$ to n . when receive $\langle CUPDATE-REPLY, ts \rangle$ from node n if $CLOG(fid).ts = ts$ $CLOG(fid) \leftarrow CLOG(fid) \setminus \{n\}$ Remove fid from $CLOG$ when $CLOG(fid)$ becomes empty
when receive $\langle CUPDATE, r, pDirs \rangle$ <i>r</i> : Replica // New replica contents. $pDirs: \mathbb{P} \langle FileID, \mathbb{P} NodeID \rangle$ // Name and location of parent dirs. <hr/> for $\langle pfid, ppeers \rangle \in pDirs$ CreateReplica($pfid, ppeers$) ResurrectDirectory($pfid$) UpdateReplica(r) send $\langle CUPDATE-REPLY, r.ts \rangle$

FIG. 18

Replacement Sheet

12/22

proc IssueUpdate

r: **Replica** // The replica of the file

del: $\mathbb{P} \langle \text{FileID}, \text{String} \rangle$ // Backpointers deleted from the replica.

if *r.fid* $\in \text{dom}(\text{ULOG})$ **then**

del $\leftarrow del \cup \text{ULOG}(r.fid)$

ULOG $\leftarrow \text{ULOG} \oplus \{r.fid \mapsto del\}$

proc ProcessUpdate // Called periodically in the background.

for (*fid* $\mapsto del$) $\in \text{ULOG}$

r $\leftarrow \text{DISK}(fid)$ // See (12).

for *pfid*, *fname* $\in del \cup r.bptrs$

ResurrectDirectory(*pfid*)

d $\leftarrow \text{DISK}(pfid)$

valid = *pfid* $\in r.bptrs$ // Is this entry to be added?

new = (*fid*, *fname*) $\mapsto \text{Dentry}(\text{valid}, r.ts, r.gpeers)$

d.ents $\leftarrow ((fid, fname) \triangleleft d.ents) \cup new$

if *d.ents* has changed

d.ts $\leftarrow \text{Newtimestamp}()$

IssueCupdate(*d*)

ULOG $\leftarrow \{\}$

FIG. 19

proc CreateReplica <i>fid</i> : FileID // The ID of the file <i>peers</i> : \mathbb{P}_1 NodeID // The known set of gold peers of the file postconditions : <i>fid</i> \in dom(<i>DISK</i>)
if <i>fid</i> \in dom(<i>DISK</i>) then return send $\langle \text{SEND-CONTENTS}, fid \rangle$ to random node <i>n</i> \in <i>peers</i> Wait until receive $\langle \text{CONTENTS}, r, pDirs \rangle$ from <i>n</i> for $\langle pfid, ppeers \rangle \in pDirs$ CreateReplica(<i>pfid</i> , <i>ppeers</i>) UpdateReplica(<i>r</i>) Add edges between <i>r</i> and random existing replicas.
when receive $\langle \text{SEND-CONTENTS}, fid \rangle$ from node <i>n</i>
<i>r</i> \leftarrow <i>DISK</i> (<i>fid</i>) <i>pDirs</i> \leftarrow $\{p: r.bptrs \bullet \langle p.fid, DISK(p.fid).gpeers \rangle\}$ // See (14). send $\langle \text{CONTENTS}, r, pDirs \rangle$ to <i>n</i> .

FIG. 20

14/22

proc ResurrectDirectory*fid*: FileID**preconditions:***fid* \in dom(*DISK*)*fid* is a directory**postconditions:**IsLive(*DISK*(*fid*))*r* \leftarrow *DISK*(*fid*)**if** IsLive(*r*) **then****return**ResurrectDirectory(*r*.deadBptr.pfid)*r*.bptrs \leftarrow { *r*.deadBptr }*r*.ts \leftarrow Newtimestamp()IssueCupdate(*r*)**let** (*pfid*, *fname*) = *r*.deadBptr •*d* \leftarrow *DISK*(*pfid*)*d*.ents(\langle *fid*, *fname* \rangle) \leftarrow **Dentry**(*true*, *r*.ts, *r*.gpeers) ⁽²⁰⁾*d*.ts \leftarrow Newtimestamp()IssueCupdate(*d*)**FIG. 21**

Replacement Sheet

15/22

A	$fid_{/}$	5:A	ents= $\{\langle fid_{alice}, "alice", 12:A \rangle\}$
	fid_{alice}	12:A	ents= $\{*\langle fid_{bar}, "bar", 13:B \rangle\}$

FIG. 22

A	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, "foo", ... \rangle\}$
	fid_{alice}	6:A	ents= $\{\}$
	fid_{foo}	8:A	bptrs= $\langle fid_{/}, "foo" \rangle$
B	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, "foo", ... \rangle\}$
	fid_{bob}	7:B	ents= $\{\}$
	fid_{foo}	8:A	bptrs= $\langle fid_{/}, "foo" \rangle$

FIG. 23

A	$fid_{/}$	10:A	ents= $\{*\langle fid_{foo}, "foo", ... \rangle\}$
	fid_{alice}	11:A	ents= $\{\langle fid_{foo}, "foo1", 12:A \rangle\}$
	fid_{foo}	12:A	bptrs= $\langle fid_{alice}, "foo1" \rangle$
B	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, "foo", ... \rangle\}$
	fid_{bob}	7:B	ents= $\{\}$
	fid_{foo}	8:A	bptrs= $\langle fid_{/}, "foo" \rangle$

FIG. 24

A	$fid_{/}$	10:A	ents= $\{*\langle fid_{foo}, "foo", ... \rangle\}$
	fid_{alice}	11:A	ents= $\{\langle fid_{foo}, "foo1", 12:A \rangle\}$
	fid_{foo}	12:A	bptrs= $\langle fid_{alice}, "foo1" \rangle$
B	$fid_{/}$	9:B	ents= $\{*\langle fid_{foo}, "foo", ... \rangle\}$
	fid_{bob}	11:B	ents= $\{\langle fid_{foo}, "foo2", 12:B \rangle\}$
	fid_{foo}	12:B	bptrs= $\langle fid_{bob}, "foo2" \rangle$

FIG. 25

Replacement Sheet

16/22

A	<i>fid</i> /	10:A	ents={* <i><fid_{foo}</i> , "foo", ...>}
	<i>fid_{alice}</i>	14:A	ents={* <i><fid_{foo}</i> , "foo1", 12:B>}
	<i>fid_{bob}</i>	11:B	ents={ <i><fid_{foo}</i> , "foo2", 12:B>}
	<i>fid_{foo}</i>	12:B	bptrs= <i><fid_{bob}</i> , "foo2">
B	<i>fid</i> /	9:B	ents={* <i><fid_{foo}</i> , "foo", ...>}
	<i>fid_{bob}</i>	11:B	ents={ <i><fid_{foo}</i> , "foo2", 12:B>}
	<i>fid_{foo}</i>	12:B	bptrs= <i><fid_{bob}</i> , "foo2">

FIG. 26

A	<i>fid</i> /	10:A	ents={* <i><fid_{foo}</i> , "foo", ...>}
	<i>fid_{alice}</i>	14:A	ents={* <i><fid_{foo}</i> , "foo1", 12:B>}
	<i>fid_{bob}</i>	11:B	ents={ <i><fid_{foo}</i> , "foo2", 12:B>}
	<i>fid_{foo}</i>	12:B	bptrs= <i><fid_{bob}</i> , "foo2">
B	<i>fid</i> /	10:A	ents={* <i><fid_{foo}</i> , "foo", ...>}
	<i>fid_{bob}</i>	11:B	ents={ <i><fid_{foo}</i> , "foo2", 12:B>}
	<i>fid_{foo}</i>	12:B	bptrs= <i><fid_{bob}</i> , "foo2">

FIG. 27

A	<i>fid</i> /	5:A	ents={ <i><fid_{foo}</i> , "foo", 6:A>}
	<i>fid_{foo}</i>	6:A	bptrs={ <i><fid_{foo}</i> , "foo">}
B	<i>fid</i> /	5:A	ents={ <i><fid_{foo}</i> , "foo", 6:A>}
	<i>fid_{foo}</i>	6:A	bptrs={ <i><fid_{foo}</i> , "foo">}

FIG. 28

A	<i>fid</i> /	10:A	ents={* <i><fid_{foo}</i> , "foo", 11:A>}
	<i>fid_{foo}</i>	11:A	bptrs={}
B	<i>fid</i> /	5:A	ents={ <i><fid_{foo}</i> , "foo", 6:A>}
	<i>fid_{foo}</i>	6:A	bptrs={ <i><fid_{foo}</i> , "foo">}

FIG. 29

Replacement Sheet

17/22

A	$fid_{/}$ fid_{foo}	10:A 11:A	ents= $\{*\langle fid_{foo}, "foo", 11:A \rangle\}$ bptrs= $\{\}$
B	$fid_{/}$ fid_{foo}	5:A 11:B	ents= $\{\langle fid_{foo}, "foo", 6:A \rangle\}$ bptrs= $\{\langle fid_{foo}, "foo" \rangle\}$

FIG. 30

A	$fid_{/}$ fid_{foo}	12:A 11:B	ents= $\{\langle fid_{foo}, "foo", 11:B \rangle\}$ bptrs= $\{\langle fid_{foo}, "foo" \rangle\}$
B	$fid_{/}$ fid_{foo}	12:A 11:B	ents= $\{\langle fid_{foo}, "foo", 11:B \rangle\}$ bptrs= $\{\langle fid_{foo}, "foo" \rangle\}$

FIG. 31

A	$fid_{/}$ fid_{foo}	12:B 11:A	ents= $\{*\langle fid_{foo}, "foo", 11:A \rangle\}$ bptrs= $\{\}$
B	$fid_{/}$ fid_{foo}	12:B 11:A	ents= $\{*\langle fid_{foo}, "foo", 11:A \rangle\}$ bptrs= $\{\}$

FIG. 32

A	$fid_{/}$ fid_{foo}	5:A 6:A	ents= $\{\langle fid_{foo}, "foo", 6:A \rangle\}$ ents= $\{\}$, bptrs= $\{\langle fid_{/}, "foo" \rangle\}$
B	$fid_{/}$ fid_{foo}	5:A 6:A	ents= $\{\langle fid_{foo}, "foo", 6:A \rangle\}$ ents= $\{\}$, bptrs= $\{\langle fid_{/}, "foo" \rangle\}$

FIG. 33

A	$fid_{/}$ fid_{foo} fid_{bar}	5:A 10:A 11:A	ents= $\{\langle fid_{foo}, "foo", 6:A \rangle\}$ ents= $\{\langle fid_{bar}, "bar", 11:A \rangle\}$, bptrs= $\{\langle fid_{/}, "foo" \rangle\}$ bptrs= $\{\langle fid_{foo}, "bar" \rangle\}$
B	$fid_{/}$ fid_{foo}	5:A 6:A	ents= $\{\langle fid_{foo}, "foo", 6:A \rangle\}$ ents= $\{\}$, bptrs= $\{\langle fid_{/}, "foo" \rangle\}$

FIG. 34

Replacement Sheet

18/22

A	$fid_{/}$	5:A	ents= $\{\langle fid_{foo}, "foo", 6:A \rangle\}$
	fid_{foo}	10:A	ents= $\{\langle fid_{bar}, "bar", 11:A \rangle\}$, bptrs= $\{\langle fid_{/}, "foo" \rangle\}$
	fid_{bar}	11:A	bptrs= $\{\langle fid_{foo}, "bar" \rangle\}$
B	$fid_{/}$	8:B	ents= $\{*\langle fid_{foo}, "foo", 10:B \rangle\}$
	fid_{foo}	10:B	ents= $\{\}$, bptrs= $\{\}$

FIG. 35

A	$fid_{/}$	8:B	ents= $\{*\langle fid_{dir}, "dir", 10:B \rangle\}$
	fid_{dir}	10:B	ents= $\{\}$, bptrs= $\{\}$
	fid_{foo}	11:A	bptrs= $\{\langle fid_{dir}, "foo" \rangle\}$
B	$fid_{/}$	8:B	ents= $\{*\langle fid_{dir}, "dir", 10:B \rangle\}$
	fid_{dir}	10:B	ents= $\{\}$, bptrs= $\{\}$

FIG. 36

A	$fid_{/}$	12:A	ents= $\{\langle fid_{dir}, "dir", 13:A \rangle\}$
	fid_{dir}	13:A	ents= $\{\langle fid_{foo}, "foo", 11:A \rangle\}$
	fid_{foo}	11:A	bptrs= $\{\langle fid_{dir}, "foo" \rangle\}$
B	$fid_{/}$	12:A	ents= $\{\langle fid_{dir}, "dir", 13:A \rangle\}$
	fid_{dir}	13:A	ents= $\{\langle fid_{foo}, "foo", 11:A \rangle\}$

FIG. 37

A	$fid_{/}$	8:B	ents= $\{*\langle fid_{dir}, "dir", 10:B \rangle\}$
	fid_{dir}	10:A	ents= $\{\langle fid_{foo}, "foo", 11:A \rangle\}$, bptrs= $\{\langle fid_{/}, "dir" \rangle\}$
	fid_{foo}	11:A	bptrs= $\{\langle fid_{dir}, "foo" \rangle\}$
B	$fid_{/}$	8:B	ents= $\{*\langle fid_{dir}, "dir", 10:B \rangle\}$
	fid_{dir}	10:B	ents= $\{\}$, bptrs= $\{\}$

FIG. 38

Replacement Sheet

19/22

A	$fid_{/}$	8:B	$ents=\{*\langle fid_{dir}, "dir", 10:B\rangle\}$
	fid_{dir}	10:A	$ents=\{\langle fid_{foo}, "foo", 11:A\rangle\},$ $bptrs=\{\langle fid_{/}, "dir"\rangle\}$
	fid_{foo}	11:A	$bptrs=\{\langle fid_{dir}, "foo"\rangle\}$
B	$fid_{/}$	8:B	$ents=\{*\langle fid_{dir}, "dir", 10:B\rangle\}$
	fid_{dir}	10:A	$ents=\{\langle fid_{foo}, "foo", 11:A\rangle\},$ $bptrs=\{\langle fid_{/}, "dir"\rangle\}$

FIG. 39

A	$fid_{/}$	13:A	$ents=\{\langle fid_{dir}, "dir", 10:A\rangle\}$
	fid_{dir}	10:A	$ents=\{\langle fid_{foo}, "foo", 11:A\rangle\},$ $bptrs=\{\langle fid_{/}, "dir"\rangle\}$
	fid_{foo}	11:A	$bptrs=\{\langle fid_{dir}, "foo"\rangle\}$
B	$fid_{/}$	13:A	$ents=\{\langle fid_{dir}, "dir", 10:A\rangle\}$
	fid_{dir}	10:A	$ents=\{\langle fid_{foo}, "foo", 11:A\rangle\},$ $bptrs=\{\langle fid_{/}, "dir"\rangle\}$

FIG. 40

20/22

```

// Called every third night for every replica on the node.
proc GarbageCollection
  LiveNodes:  $\mathbb{P}_1$  NodeID // Set of live nodes.
  r: Replica // Replica to be inspected.
  EXPIRE: integer // Dead-replica expiration period, e.g., a month.

  // Remove old tombstones. Removing after EXPIRE seconds is safe
  // because we cannot receive any new update with timestamp older
  // than EXPIRE after removing r.
  if  $\neg \text{IsLive}(r)$  and r.ts < Newtimestamp - EXPIRE then
    DISK  $\leftarrow \{r.fid\} \triangleleft \text{DISK}$ 
    return

  r'  $\leftarrow \text{Deepcopy}(r)$ 

  // Remove dead entries in the directory
  if r' is a directory then
    for (key  $\mapsto$  val): r'.ents do
      if  $\neg \text{val.valid}$  and val.ts < Newtimestamp() - EXPIRE then
        r'.ents  $\leftarrow \{key\} \triangleleft r'.ents$ 
      if at least one entry has been removed from r'.ents then
        r'.ts  $\leftarrow \text{NewTimestamp}()$ 
        UpdateReplica(r)

  // If some gold peers are found dead, recreate one elsewhere
  if me  $\in r.gpeers$  and r.gpeers  $\not\subseteq \text{Livenodes}$  then
    newNodes  $\leftarrow \text{Pick } ||r.gpeers \setminus \text{Livenodes}||$  random live nodes.
    r'.gpeers  $\leftarrow r.gpeers \setminus \text{Livenodes} \cup \text{newNodes}$ 
    r'.ts  $\leftarrow \text{NewTimestamp}()$ 
    UpdateReplica(r)

  // If we find graph edges to dead nodes, re-span it.
  for n  $\in r.peers \setminus \text{Livenodes}$ 
    Add edges between r and a random replica.
    r.peers  $\leftarrow r.peers \cap \text{Livenodes}$ 

```

FIG. 41

21/22

$$\begin{aligned} & \forall f: \text{ran}(\text{DISK}) \text{ and } \text{IsLive}(f) \bullet \\ & (\langle d.\text{fid}, \text{fname} \rangle \in f.\text{bptrs} \\ & \Rightarrow d \in \text{dom}(\text{DISK}) \text{ and } \text{IsLive}(d)) \end{aligned}$$

FIG. 42

$$\begin{aligned} & \forall d: \text{ran}(\text{DISK}) \text{ and } \text{IsLive}(d) \bullet \\ & (f: \text{ran}(\text{DISK}) \text{ and } \text{IsLive}(f) \bullet \\ & (\langle d.\text{fid}, \text{fname} \rangle \mapsto \text{ent}) \in d.\text{ents} \text{ and } \text{ent.valid} \\ & \text{and } \langle d.\text{fid}, \text{fname} \rangle \notin f.\text{bptr} \\ & \Rightarrow \Diamond \langle f.\text{fid}, \text{fname} \rangle \mapsto \text{ent} \notin d.\text{ents}) \end{aligned}$$

FIG. 43

$$\begin{aligned} & \forall d: \text{ran}(\text{DISK}) \text{ and } \text{IsLive}(d) \bullet \\ & (\langle f.\text{fid}, \text{fname} \rangle \mapsto \text{ent} \in d.\text{ents} \text{ and } \text{ent.valid} \\ & \Rightarrow f: \text{ran}(\text{DISK}') \text{ and } \text{IsLive}(f)) \end{aligned}$$

FIG. 44

22/22

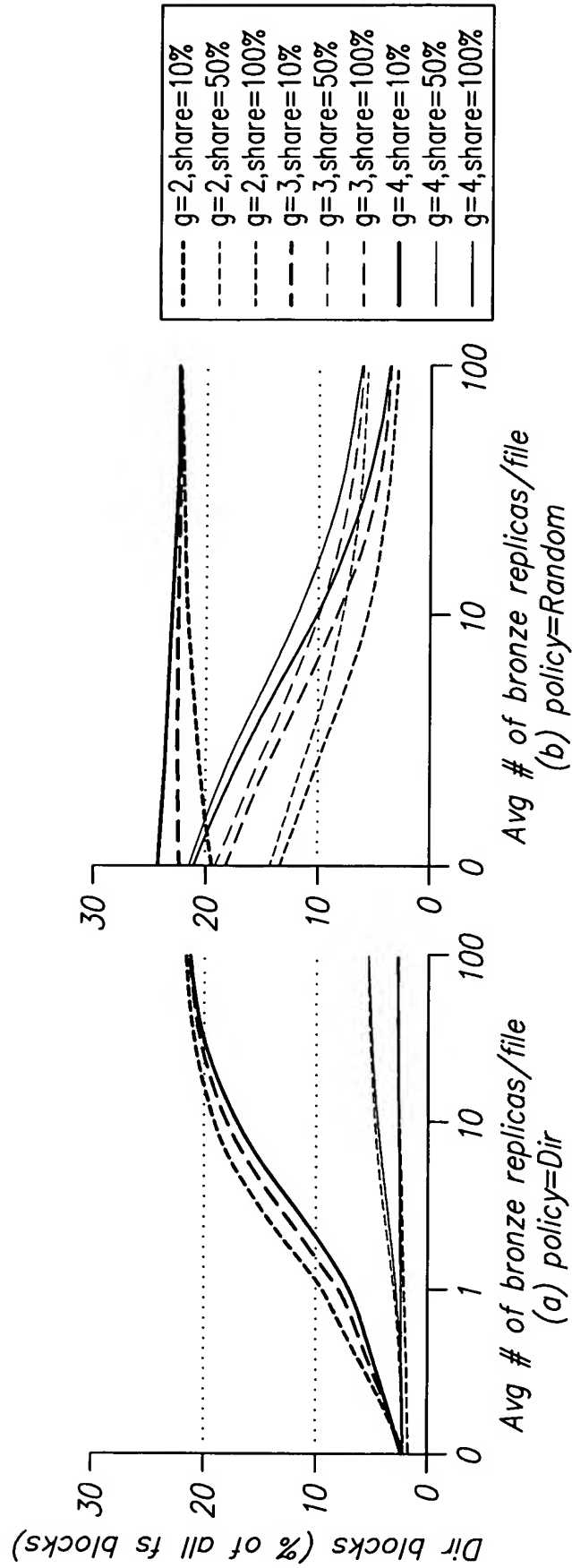


FIG. 45